

AD-A131 281

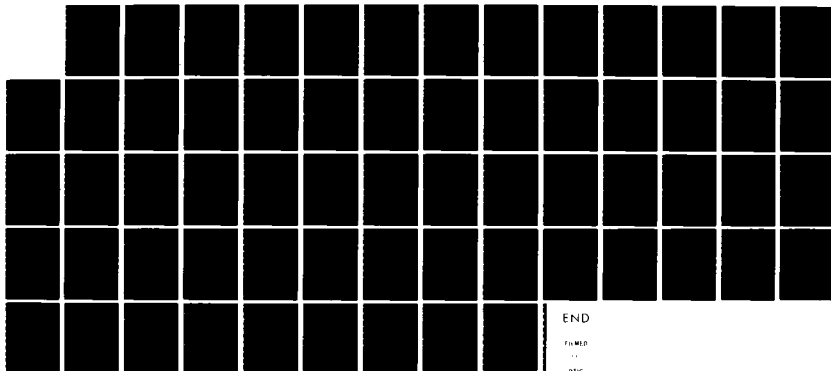
LANGUAGE PROCESSING FOR SPEECH UNDERSTANDING(U) BOLT
BERANEK AND NEWMAN INC CAMBRIDGE MA W A WOODS JUL 83
BBN-5376 N00014-77-C-0371

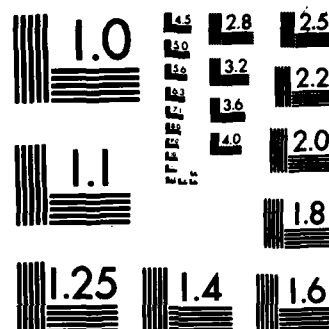
1/1

UNCLASSIFIED

F/G 5/7

NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Bolt Beranek and Newman Inc.

12 

AD A 1 3 1 2 8 1


Report No. 5376

Language Processing for Speech Understanding
Technical Report

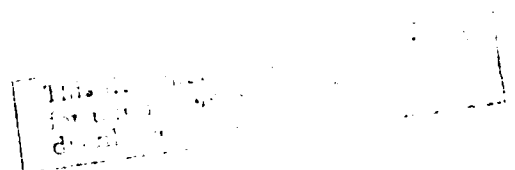
W.A. Woods

July 1983

Prepared for:
The Office of Naval Research


C. J. A

DTIC FILE COPY



83 08 08 032

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER BBN Report No. 5376	2. GOVT ACCESSION NO. AD A131 381	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) LANGUAGE PROCESSING FOR SPEECH UNDERSTANDING		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER BBN Report No. 5376
7. AUTHOR(s) W. A. Woods		8. CONTRACT OR GRANT NUMBER(s) N00014-77-C-0371
9. PERFORMING ORGANIZATION NAME AND ADDRESS Bolt Beranek and Newman Inc. 10 Moulton St. Cambridge, MA 02238		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research Department of the Navy Arlington, VA 22217		12. REPORT DATE July 1983
		13. NUMBER OF PAGES 53
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce, for sale to the general public.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Speech, speech understanding, SUR, control strategy, parsing, middle-out parsing, ATN grammars, HWIM, shortfall density scoring.		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report considers language understanding techniques and control strategies that can be applied to provide higher-level support to aid in the understanding of spoken utterances. The discussion is illustrated with concepts and examples from the BBN speech understanding system, HWIM. The HWIM system was conceived as an assistant to a travel budget manager, a system that would store information about planned and taken trips, travel budgets and their planning. The system was able to respond to commands cont'd		

DD FORM 1473
1 JAN 73

EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (cont'd.)

and answer questions spoken into a microphone, and was able to synthesize spoken responses as output. HWIM was a prototype system used to drive speech understanding research. It used a phonetic-based approach, with no speaker training, a large vocabulary, and a relatively unconstraining English grammar. Discussed here is the control structure of the HWIM and the parsing algorithm used to parse sentences from the middle-out, using an ATN grammar.

Unclassified.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Report No. 5376

LANGUAGE PROCESSING FOR SPEECH UNDERSTANDING

Technical Report

July 1983

Principal Investigator:
William A. Woods
(617) 497-3361

Prepared by:

Bolt Beranek and Newman Inc.
10 Moulton Street
Cambridge, Massachusetts 02238

Prepared for:

The Office of Naval Research

This research was supported by the Office of Naval Research under Contract No. N00014-77-C-0371. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Office of Naval Research or the U.S. Government.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Date	
List	
A	



TABLE OF CONTENTS

	Page
1. THE PROBLEM OF SPEECH UNDERSTANDING	1
1.1 Knowledge Components for Speech Understanding	3
1.2 Putting it all Together	4
2. CONTROL STRATEGIES FOR SPEECH PERCEPTION	7
2.1 Island-Driven Strategies	9
2.2 Priority Scoring	11
2.3 Shortfall Scoring	12
2.4 Density Scoring	14
2.5 Shortfall Density Scoring	15
2.6 Focus of Attention by a MAXSEG Profile	15
2.7 Admissible versus Inadmissible Strategies	16
3. ATN GRAMMARS	19
3.1 HWIM's ATN Grammar Notation	23
3.2 A Middle-Out View of an ATN Grammar	24
3.3 Cascaded ATN's	28
3.4 Lexical Retrieval as an ATN	30
3.5 Benefits of CATN's	33

4. MIDDLE OUT PARSING WITH ATN'S	35
4.1 State Relations Used in the HWIM Parser	37
4.2 Paths	39
4.3 The Parser	40
4.4 The Principal Parser Functions	41
4.5 Starting an Island	42
4.6 Processing an Arc	42
4.7 Connecting a New Word to an Island	43
4.8 Processing the Completion Queue	45
4.9 Completing a Constituent	46
4.10 Making Predictions	47
5. MIDDLE-OUT PARSING WITH CATN'S	49

LIST OF FIGURES

FIG. 1.	MEASURING SHORTFALL FROM A MAXSEG PROFILE	13
FIG. 2.	AN EXAMPLE OF AN ATN GRAMMAR	21
FIG. 3.	A BNF SPECIFICATION OF HWIM'S ATN GRAMMAR NOTATION	24
FIG. 4.	A LEXICAL DECISION TREE	31
FIG. 5.	A PROTOTYPICAL ISLAND CONFIGURATION IS SHAPED LIKE A STILE	36

1. THE PROBLEM OF SPEECH UNDERSTANDING

A fundamental characteristic of the speech understanding task, above and beyond the difficulty of acoustic phonetic recognition, is that there is generally not enough information in the acoustic signal alone to determine the phonetic content of the message. Acoustic cues are sufficient to constrain the possible identities of a given sound but frequently not sufficient to uniquely determine its identity. For example, it may be possible to determine that a given sound is a weak fricative but not whether it is an "f" or "th". The subtle acoustic cues which distinguish these sounds may be unreliable or unavailable either due to noise or distortion in the communication channel or to errors in pronunciation or lack of careful articulation on the part of the speaker.

There is ample evidence from human perceptual experiments [Wanner, 1973] that this inability is not just a limitation of mechanical signal analysis and computerized acoustic-phonetic analysis, but rather a fundamental characteristic of human speech. The human listener is highly skilled at using his "common sense" knowledge of the vocabulary and syntax of his language and his expectations for what his speaker might say to compensate for small errors in pronunciation, and possibly for this reason, people generally don't speak any more distinctly than is necessary to make themselves understood.

Thus, one cannot expect to achieve continuous speech understanding without making use of the higher level constraints imposed on an utterance by the vocabulary and syntax of the language and the knowledge of what makes sense. In this chapter I will consider language understanding techniques and control

strategies that can be applied to provide higher level support from these sources to aid in the understanding of spoken utterances.

The discussion will be illustrated with concepts and examples from the BBN speech understanding system, HWIM (which stands for "Hear What I Mean") [10, 8]. The HWIM system was conceived as an assistant to a travel budget manager, a system that would store information about planned and taken trips, travel budgets and their status, plane fares and per diems, and other information important to planning. This task was chosen as a small and easily comprehensible version of a generalized management problem. The system was able to respond to commands and answer questions spoken into a microphone and was able to synthesize spoken responses as output.

HWIM was a prototype system used to drive speech understanding research, and during the life of the project reached a level of performance where it could understand approximately 50% of a set of new test utterances, using a phonetic-based approach, with no speaker training, and a relatively unconstraining English grammar. Although the system was never fully debugged and tuned and its acoustic phonetic knowledge was incomplete, the success rate for complete understanding of (52%) of a 400 word system (branching ratio 67) fell only 8 percentage points (To 44%) when tested with no further tuning on a 1000 word vocabulary (branching ratio 196). This indicates a substantial robustness in the method.

1.1 Knowledge Components for Speech Understanding

One can identify the following conceptually distinct sources of knowledge as important in determining the interpretation of a spoken utterance:

a) Segmentation and Labeling

A process of detecting acoustic-phonetic events in the speech signal and characterizing the nature of the individual segments of the signal.

b) Lexical Retrieval

A process of retrieving candidate words from the lexicon that are acoustically similar to the labeled segments.

c) Word Matching

A process of determining some measure of the goodness of a word hypothesis at a given point in the speech signal.

d) Syntax

The ability to determine if a given sequence of words is a possible subpart of a grammatical sentence and to predict possible continuations for such sentence fragments.

e) Semantics

The ability to determine if a given hypothesized sentence is meaningful or nonsensical (in addition to being grammatical).

f) Pragmatics

The ability to determine if a sentence is appropriate to the context in which it is uttered, given knowledge of the particular speaker, the task he is trying to accomplish, and what has been said previously in the discourse.

g) Prosodics

The ability to use cues such as intonation and rhythm to predict the possible syntactic structure of an utterance or to confirm or reject a proposed syntactic structure.

In addition to these sources of knowledge, there is a major issue of control, i.e., the framework and algorithms for making decisions about which possible fragmentary hypotheses to rule out, which ones to pursue further by trying to find compatible interpretations of adjacent portions of the utterance, when to return to a previously rejected hypothesis in light of new information, etc.

1.2 Putting it all Together

It is one thing to say that all of the above processes and sources of knowledge must interact in the understanding of continuous speech. It is another thing to know how to do it. A variety of different approaches have been explored. Generally, they fall into two classes -- top down, or syntax driven, and bottom-up, or lexically driven. Some systems attempt to take a grammar of possible things to say and use it to predict possible words which the system then attempts to verify against the input. Other systems attempt to recognize words either at the phoneme or syllable level and then use dictionary entries for these words to drive a syntactic component. In both cases, one component is used as a generator of possible hypotheses and the other as a filter.

In general, the syntax driven approaches are successful only

for highly constrained applications where the system has a good chance of predicting a small number of choices for the next word at each point in the input. For less restricted situations, only lexically driven approaches have been moderately successful, although this requires some clever lexical retrieval techniques to avoid having to match every possible vocabulary item against the input.

A major problem is how to integrate the different sources of knowledge in such a way as to exploit their interaction. For example, the HWIM system, for lack of any more suitable integrative framework, combined syntactic, semantic, and pragmatic information into a single "pragmatic" ATN grammar, while keeping separate components for the phonetic and lexical levels. The CMU Harpy system [6] achieved integration by compiling a finite state grammar together with lexical and phonetic information into a single network of possible phoneme sequences, each path through which corresponded to a possible sentence. Hearsay-II [5], on the other hand, created a blackboard structure to attempt to coordinate a diverse collection of multiple knowledge sources.

In what follows, I will attempt to set forth a framework in which to view the interaction of such knowledge sources and discuss some general algorithms and techniques for using higher level knowledge for speech understanding.

Bolt Beranek and Newman Inc.

Report No. 5376

2. CONTROL STRATEGIES FOR SPEECH PERCEPTION

Speech understanding is a special case of a general class of perceptual processes. Perception can be viewed as the process of forming a believable coherent hypothesis which can account for some or all of one's sensory stimuli. Although this process is generally subconscious and one is not aware of any substeps, it can be thought of computationally as a derivation of a comprehensive hypothesis that is arrived at by successive refinement and extension of partial hypothesis until a best complete hypothesis is found. I will refer to this successive refinement as incremental perception.

In general, a perceptual system must incorporate some basic epistemological assumptions about the things which it can perceive and the rules governing their assembly. That is, the object perceived is generally a compound object, constructed from members of a finite set of elementary constituents according to known well-formedness rules. The well-formedness rules can be used to reject impossible interpretations of the input stimuli, and may also be useable to predict other constituents that could be present if a given partial hypothesis is correct. The elementary constituents, as well as the relationships among them that are invoked in the well-formedness rules, must be directly perceptible. In the case of speech understanding, the directly perceptible elements are the basic speech phonemes (or perhaps their features), and the well-formedness rules are the rules governing their formation into words, phrases, sentences, and coherent discourse.

In this section, we will be concerned with strategies governing the formation and refinement of partial hypotheses

about the identity of a speech utterance. We assume a system that contains the following components:

- a) A Lexical Retrieval component that can find the k best matching words in any region of an utterance subject to certain constraints and can be recalled to continue enumerating word matches in decreasing order of goodness (where possible constraints include anchoring the left or right end of the word to particular points in the utterance or to particular adjacent word matches). We assume that this component is interfaced to appropriate signal processing, acoustic-phonetic and phonological analysis components, as in HWIM, and that it assigns a "quality" score to each word match reflecting the goodness of the match.
- b) A Linguistic component that, given any sequence of words, can determine whether that sequence can be parsed as a possible initial, final, or internal subsequence of a syntactically correct and semantically and pragmatically appropriate utterance, and can propose compatible classes of words at each end of such a sequence.

A control strategy for such a system must answer questions such as:

- a) At which points in the utterance to call the Lexical Retrieval component, and when,
- b) What number of words to ask for,
- c) When to give subsequences of the results to the Linguistic component, and
- d) When to recall the Lexical Retrieval component to continue enumerating words at a given point.

The goal of the control strategy is to discover the best scoring sequence of words that covers the entire utterance and is acceptable to the Linguistic component. We will consider here a particular class of control strategies which we refer to as "island-driven".

2.1 Island-Driven Strategies

In an island-driven control strategy, partial hypotheses about the possible identity of the utterance are formed around initial "seed" words somewhere in the utterance and are grown into larger and larger "island" hypotheses by the addition of words to one or the other end of the island. Occasionally, two islands may "collide" by proposing and discovering the same word in the gap between them and may then be combined into a single larger island.

Each island hypothesis is evaluated by the Lexical Retrieval component to determine its degree of match with the acoustic evidence and is checked for syntactic, semantic, and pragmatic consistency by the Linguistic component. We will refer to a partial hypothesis that has been so evaluated and checked for consistency as a "theory". The strategies that we will consider operate by successively processing "events" on an event queue, where events correspond to suspended or dormant processes that may result in the creation of theories.

The general algorithm operates as follows:

- (1) An initial scan of the utterance is performed by the Lexical Retrieval component to discover the n best matching words anywhere in the utterance according to some criterion of "best" and for some value n . An initial seed event is created for each such word and placed on the event queue. In addition, one or more continuation events, which can be processed to continue the enumeration of successively lower scoring words (regardless of position in the utterance), is created and placed on the queue. Each seed event is assigned a priority score (derived, in one of

several ways to be described shortly, from the quality score that the Lexical Retrieval component gave it). Each continuation event is assigned a priority score that can be guaranteed to bound the priority score of any word that can be generated by that event (e.g., derived from the score of the last word enumerated prior to the continuation). The events are ordered on the event queue by their priority scores and are processed in order of priority.

(2) The highest priority event is selected for processing. This consists of (i) creating the corresponding theory (a one-word theory in the case of a seed event), (ii) calling the Linguistic component to check the consistency of the theory and to make predictions for words and/or word classes that can occur adjacent to it, at each end of the theory, (iii) calling the Lexical Retrieval component to enumerate the k best matching words satisfying these predictions at each end of the theory, and (iv) generating a "word" event for each such word found. A word event is an event that will add one word to a theory to create a larger theory. Continuation events are also created that will continue the enumeration of successively lower scoring words adjacent to the theory. If island-collision is permitted as an operation (island collision is a feature than can be permitted or not), then each word event generated is checked against an island table to see if the same word (at the same position in the input) has been proposed and found in the other direction by some theory. If so, an "island-collision" event is created that will combine the new word and the two theories on either side of it. Both word and island-collision events are assigned priority scores derived from the quality scores of the words that they contain and are inserted into the event queue according to their priorities.

(3) Step 2 is repeated until a theory is discovered that spans the entire utterance and is syntactically, semantically, and pragmatically acceptable as a complete sentence.

Although the basic island-driven strategy is presented here as involving an initial scan of the entire utterance before beginning the processing of events, there is nothing to prevent an implementation from dovetailing this initial scan with the event processing so that, for example, event processing on the early portions of an utterance could begin before the entire utterance had been heard.

2.2 Priority Scoring

The score assigned to a theory by the summation of lexical retrieval scores we refer to as the quality score of the theory. One can distinguish this from a possibly separate score called the priority score, which is used to rank order events on the event queue to determine the order in which they are to be done. A desirable property for a priority score is a guarantee that the first complete theory found will be the best scoring one that can be found. Using the quality scores directly as priority scores does not ordinarily provide such a guarantee. That is, a straightforward "best-first" search strategy does not guarantee discovery of the best overall hypothesis.

An algorithm that is guaranteed in this way to find the best entity in some search space is said to be admissible. For speech understanding applications, admissibility is a desirable property, but not necessarily essential if the cost of its attainment is too great. In this section we will discuss several

priority scores, derived from but not identical with the quality score, that result in admissible algorithms. The first measures the difference between the quality score for a theory and an upperbound on the possible quality for any theory covering the same portion of the utterance. This is called the shortfall score. Two other priority scores are obtained by dividing either the quality score or the shortfall score by the time duration on the island to give quality density and shortfall density scoring, respectively.

2.3 Shortfall Scoring

Shortfall scoring measures the amount by which the score of a theory falls below an upper bound on the possible score that could be achieved on the same region. When shortfall scoring is being used, a MAXSEG profile is constructed having the property that the score of a word match between boundaries i and j will be less than or equal to the area under the MAXSEG profile from i to j (call this latter the MAXSCORE for the region from i to j). The shortfall score for a theory is then computed as the sum over all the word matches in the theory of the difference between the score of the word match and the MAXSCORE for the same region. The relationship of the MAXSEG profile to the actual score of a theory is illustrated in Figure 1.

The theoretical characteristics of the shortfall scoring algorithm are that if the words are returned by the Lexical Retrieval component in shortfall order and events are processed in order of increasing magnitude of shortfall (plus a few other assumptions, documented in [14]), then the first complete

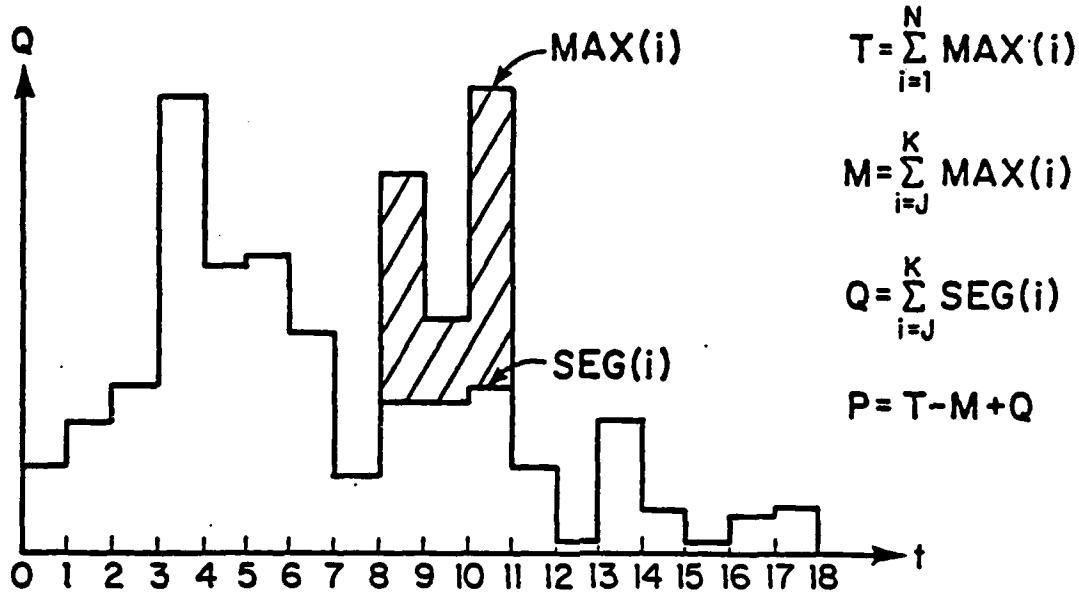


FIG. 1. MEASURING SHORTFALL FROM A MAXSEG PROFILE

spanning interpretation found will be the best scoring interpretation that can be found by any strategy (modulo ties). For any quality metric that is additive, shortfall scoring is admissible without searching the entire space of possible word sequences. The proof of admissibility depends only on the fact that when the first complete spanning theory is found, all other events on the queue will already have fallen below the ideal maximum score by at least as much. Thus the result does not depend on the scores being likelihood ratios, nor does it make any assumption about the nature of the grammar (e.g., that it be a finite state markov process), provided a parser exists that can

make the necessary judgements. The admissibility of the basic shortfall method also does not depend on the order of scanning the utterance -- it is true both for middle-out and for left-to-right strategies.

2.4 Density Scoring

Another type of priority scoring is density scoring. Here the score used to order the eventqueue is some basic score divided by the duration of the event. Conceptually, we can think of this scoring metric as predicting the potential score for the region not covered by a theory to be an extrapolation of the same score density already achieved. (In these terms, the shortfall strategy can be thought of as predicting that the potential score for the uncovered region will achieve the upper bound.) Unlike the shortfall scores, density scores can get worse and then get better again as new words are added to a theory. Hence the density score is not guaranteed to be an upper bound of the expected eventual score. However, it has another interesting property: in exactly those cases where it does not upperbound the eventual score, there is a word to be added somewhere else that has a better score density and whose score density does upperbound the eventual score. This arises from the property of densities that the density of two regions combined will lie between the densities that they each have. It turns out that this alone is not sufficient to guarantee admissibility, since it is still possible for the density score starting from the best correct seed to fall below that of some other less-than-optimal spanning theory before it can be extended to a complete theory itself. However, with the addition of a facility for island

collisions, the density scoring strategy working middle-out from multiple seeds has been shown to be admissible. Density scoring does not depend on any assumptions about the basic scores to which it is being applied other than that they be additive (and capable of division). Hence the density method can be applied to either the original quality score or to a shortfall score.

2.5 Shortfall Density Scoring

The shortfall density method of scoring partial hypotheses is a combination of the shortfall and density methods. Experimental comparison of the algorithms [14] suggests that the shortfall density method is superior to quality density, which is in turn superior to the shortfall method alone. The superiority of the density methods over the shortfall method can be accounted for by the excessive conservatism (over optimistic scoring of alternative hypotheses) of the shortfall method. The superiority of the combined shortfall density method can be attributed to an improved "focus of attention" strategy as follows:

2.6 Focus of Attention by a MAXSEG Profile

A major effect of prioritizing events by the shortfall from a MAXSEG profile is that the score differences in different parts of the utterance are effectively leveled out, so that events in a region of the utterance where the best word matches are not very good can hold their own against alternative interpretations in regions where there are high quality words. This promotes the refocusing of attention from a region where there may happen to

be high quality accidental word matches only slightly worse than the best, to other regions whose best word match quality may not be as great. If this were not done, then many secondary matches in the high scoring region could be considered before any theories worked their way across the low scoring regions. Thus, an apparently satisfactory and intuitively reasonable strategy for focusing attention emerges from the same method that guarantees admissibility.

Notice that in the shortfall density method, the MAXSEG profile is no longer serving the role of guaranteeing admissibility that it did in the shortfall method. In this case, the admissibility is guaranteed by the nature of densities and island collisions. Rather, in this method the MAXSEG profile is used only to provide this leveling of effort over portions of the utterance to promote the refocusing of attention. In fact, it is no longer necessary that the MAXSEG profile be an upper bound (although there are undesirable effects when the shortfall density goes negative).

2.7 Admissible versus Inadmissible Strategies

The admissible strategies discussed above are only some of the control strategy options implemented in the HWIM speech understanding system. In addition there are a large number of strategy variations that result in deliberately inadmissible strategies, including strictly left-to-right density strategies and "hybrid" strategies that start near the left end of an utterance and work left to the end and then left-to-right across the rest of the utterance. For reasons of time and resource

limitations, the final test run of the HWIM system was made using one of the inadmissible hybrid left-to-right strategies. Subsequently, a much smaller experiment was run to compare various control strategies on a set of ten utterances chosen at random from the larger set. Although this sample is much too small to be relied on, the results are nevertheless suggestive. For two comparable experiments using the best left-to-right method (left-hybrid shortfall density) and the best nearly admissible method (shortfall density with ghosts, island collisions, and direction preference), both with a resource limitation of 100 theories, the inadmissible left-hybrid strategy found the best (and in these cases the correct) interpretation within the resource limitation in 6 of the 10 cases and misinterpreted two additional utterances with no indication to distinguish them from the other 6. The shortfall density strategy found 5 correct interpretations (not a significant difference for this size sample) and rejected the others.

If this left-hybrid strategy were used in an actual application with comparable degrees of acoustic degradation (e.g., due to a noisy environment), the system would claim to understand most of its utterances, but would actually misunderstand a significant fraction of those due to failure to find the best interpretation. The shortfall density strategy, on the other hand, would only misunderstand an utterance if its correct interpretation actually had a lower score than the one it found (hopefully a negligible fraction of the cases).

The middle-out shortfall density algorithm in the above experiments expanded only 50% more theories (and incidentally used only 30% more cpu time) than did the left-hybrid strategy. Although as we said before, this test set is much too small to

draw firm conclusions, the success rate of the two methods are not much different, except that the middle-out method is clearly less likely to make an incorrect interpretation. If one considers proposals to improve the performance of inadmissible strategies by having them continue to search for additional interpretations after the first one is found, then the time difference shown above could easily be reversed and there would still be no guarantee that the interpretation found would be the best one.

3. ATN GRAMMARS

Having considered the issue of control strategies for using higher level knowledge in speech understanding, let us now turn to the problem of representing and using that knowledge to make the judgements required. The principal device that I will present for this purpose is the concept of an Augmented Transition Network (ATN) grammar. ATN grammars, as presented in [9], are a form of pushdown store automata, augmented to carry a set of register contents in addition to state and stack information and to permit arbitrary computational tests and actions associated with the state transitions. Conceptually, an ATN consists of a network of states representing partial states of knowledge that arise in the course of parsing a sentence. States are connected by arcs indicating kinds of constituents that can cause transitions from one state to another. The states in the network can be conceptually divided into "levels" corresponding to the different constituents that can be recognized. Each such level has a start state and one or more final states, and behaves as a recognition automaton for its particular kind of constituent.

Transitions between states are of three basic types, indicated by three different types of arc. A WRD (or CAT) transition corresponds to the consumption of a single element from the input string, a JUMP transition corresponds to a transition from one state to another without consuming any of the input string, and a PUSH transition corresponds to the consumption of a phrase parsed by a subordinate invocation of some level of the network to recognize a constituent.

ATN's have the advantage of being a class of automata into

which ordinary context-free phrase structure and "augmented" phrase structure grammars have a straightforward embedding, but which permit various transformations to be performed to produce grammars that can be more efficient than the original. Such transformations can reduce the number of states or arcs in the grammar or can reduce the number of alternative hypotheses that need to be explicitly considered during parsing. Both kinds of efficiency result from a principle that I have called "factoring", which amounts to merging common parts of alternative paths in order to reduce the number of alternative combinations explicitly enumerated. Conceptual factoring results from merging common parts of the grammar to make the grammar as compact as possible, while hypothesis factoring results from arranging the grammar so as to merge common parts of hypotheses that will be enumerated at parse time (see [9] for further discussion).

Figure 2 illustrates a small fragment of an ATN for recognizing basic sentences of English. This grammar compactly represents the information that would be specified by the following infinite set of context free grammar rules:

```
S -> NP Vint
S -> NP Vtr NP
S -> NP Vind NP NP
S -> NP AUX Vint
S -> NP AUX Vtr NP
S -> NP AUX Vind NP NP
S -> AUX NP Vint
S -> AUX NP Vtr NP
S -> AUX NP Vind NP NP
S -> NP Vint PP
S -> NP Vtr NP PP
S -> NP Vind NP NP PP
.
.
.
S -> NP Vint PP PP
.
```

.
 S -> NP Vint PP PP PP
 .
 .
 .

(where NP stands for Noun Phrase, PP for Prepositional Phrase, Vint for an intransitive verb, Vtr for a transitive verb, Vind for a verb that takes indirect objects, and AUX for an auxiliary verb such as "is" or "does").

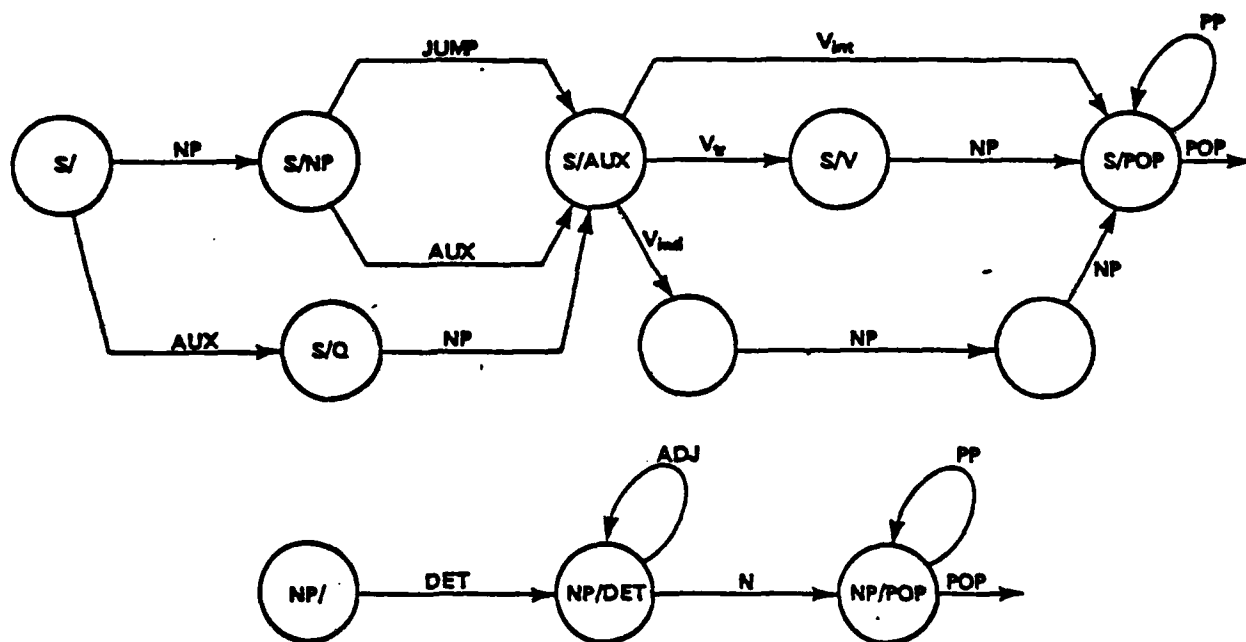


FIG. 2. AN EXAMPLE OF AN ATN GRAMMAR

ATN grammars are very effective for specifying complex grammars of natural language as well as for a variety of other structured entities. One can think of them as a class of

abstract perceptual automata for recognizing structured sequences of elements.

A state in an ATN can be thought of dually as a concise representation of a set of alternative possible sequences of elements leading to it from the left or as a concise prediction of a set of possible sequences of elements to be found to the right. Alternatively, it can be thought of in a right-to-left mode as a concise representation of a set of possible sequences of elements found to the right and a prediction of possible sequences to be found to the left. The reification of these states as concrete entities that can be used to represent partial states of knowledge and prediction during parsing is one of the major contributions of ATN grammars to the theory and practice of natural language understanding. They are even more important in representing states of partial knowledge in the course of speech understanding.

In addition to concisely specifying alternative sequences of constituents, ATN grammars serve as a conceptual map of possible sentence structures and a framework on which to hang information about constraints that apply between separate constituents of a phrase and the output structure that the grammar would like to assign to a phrase. This is done through conditions and structure-building actions associated with the arcs of the ATN. These conditions and actions operate on the constituent being accepted and a set of registers that can hold arbitrary information picked up elsewhere in the parse. Although in the original ATN formulation, these were presented as arbitrary LISP procedures to be executed in the context of a left-to-right parse of a sentence, they can be viewed as general constraints to be applied and generalized specifications of intended structure assignments.

3.1 HWIM's ATN Grammar Notation

The ATN grammars used by HWIM use five different arc types: PUSH, POP, WRD, CAT and JUMP.

A PUSH arc essentially "consumes" a phrase of a specified type (e.g., a noun phrase) by causing a an invocation of a subordinate transition network corresponding to the desired type of phrase. If the lower level transition network can successfully accept the next segment of the input string, it is exited by a POP arc, and processing will continue in the transition network containing the invoking PUSH arc. Each final state in a transition network will contain a POP arc that signifies successful completion of its level of the network and indicates what structure is to be returned for use by its calling network.

WRD and CAT arcs are the only terminal consuming (i.e., word consuming) arcs in the grammar. That is, they are the only arcs that advance the "input pointer" that marks the current position in the input string. (This pointer may move during a PUSH arc, but only as a result of WRD and CAT arcs taken in the lower network.) WRD and CAT arcs differ only in the way they express the set of terminals they can consume. A WRD arc specifies its terminals explicitly and exhaustively, while a CAT arc specifies them implicitly via the syntactic-semantic category to which they must belong.

A JUMP arc causes a transition between states, but does not consume a terminal in doing so and hence does not advance the input pointer.

Figure 3 shows the notation used for describing a HWIM ATN grammar; it is similar to most other ATN formalisms, except that conditions on arcs are expressed in terms of an action (VERIFY <condition>) and actions can be embedded in SCOPE statements to indicate left context dependencies (see section 3.2).

```

<ATN> -> (<state> <state>*)
<state> -> (<state-name><arc><arc>*)
<arc> -> <W> | <B> | <C> | <J>

<W> -> (CAT <category-name> <action>* (TO <state-name>))
      (WRD <terminal "word"> <action>* (TO <state-name>))

<B> -> (PUSH <state-name> <action>* (TO <state-name>))
<C> -> (POP <form> <action>*)
<J> -> (JUMP <state-name> <action>*)

<action> -> <action1> |
      (SCOPE <scope-spec> <action1> <action1>*)

<action1> -> (VERIFY <action1>) |
      <register-getting-action> |
      <structure-building-action> |
      <testing-action>

<scope-spec> -> (<state-name> <state-name>*) |
      NIL | T

```

FIG. 3. A BNF SPECIFICATION OF HWIM'S ATN GRAMMAR NOTATION

3.2 A Middle-Out View of an ATN Grammar

HWIM's island-driven control strategy requires a parser that can begin in the middle of an utterance and extend an island in either direction. This entails an ability to parse right-to-left as well as left-to-right and to do so with an incomplete context

in the other direction. Since ATN grammars are normally conceived as parsing automata that proceed in a left to right manner from the beginning of a sentence, setting and testing registers as they go, it is not immediately obvious that they could be used by such a parser. In this section we will discuss how this can be done.

In one view of ATN grammars, a state is viewed merely as a bundle of arcs, and an arc is merely a component of its begin-state. While this conceptualization is adequate for left to right parsing, for HWIM's parser, it is more useful to think of an arc as a connection between two states that can be traversed in either direction. An arc, then, is associated with a left state, a right state, a type (WRD, CAT, etc.), a label (NP, AUX, Vint, etc.), a set of context-free actions that can be done when the arc is first encountered regardless of the direction or context, and a set of context sensitive actions, which will be deferred, if necessary, until adequate left context is available. A state, then, has two associated collections of arcs, one set leading to the left and the other leading to the right.

The major difficulty in using an ATN grammar for middle-out parsing stems from the way that conditions and actions on the arcs use registers. During normal left-to-right processing in a standard ATN parser, actions call for both accessing and changing the contents of registers that have been set by arcs to their left in the grammar. We will say that an arc action in the grammar has a left dependency when it either requires the value of a register that is set somewhere to the left or changes the value of a register that is used somewhere to the left. If such an action is executed on an arc in the middle of a sentence without having processed a suitable left context problems will

arise. In the first case, executing the action without the left context could not produce the correct effect, while in the second case, executing it prematurely in a right-to-left parsing would cause the later execution of the arc action to the left to get the wrong value. Such arc actions are referred to as context sensitive. Fortunately, fewer than half of the actions in the HWIM ATN grammars turn out to have such dependencies.

This problem is solved in HWIM's grammars by providing all context sensitive actions with a scope specification that indicates what left context is necessary before the action can be executed. By analysis of the paths through the grammar, it is possible to determine, for each context sensitive arc action, a set of states having the property that the action can be safely done if its execution in a right-to-left parse is delayed until the parse has passed through one of those states. We refer to this set of states as the scope of the action, an indication of how far left in the grammar its left-to-right dependency extends. The HWIM parser interprets scope specifications on arc actions by saving the action with its local context until its scope is satisfied (if it is not already satisfied when the action is first considered).

This scoping mechanism allows a grammar that was created from a left-to-right viewpoint to be used in middle-out parsing with very little modification. It is only necessary to add appropriate scope specifications to the context-sensitive conditions and actions. To the grammar writer, the ATN can remain basically a left to right machine; its arc actions can be written almost as if the parser were operating only left to right. The grammar has actions on arcs where they should be executed if the entire appropriate left context were set. As in

standard left-to-right grammars, in no case is it ever necessary to worry about the right context. This is not necessarily the best way to represent bidirectional ATN grammars, but it works. It would be possible to construct an algorithm to compute the scopes automatically if the dependencies of the arc actions were explicitly marked, but for reasons of expediency no such facility was implemented in HWIM. Rather, the scope annotations were created by hand.

The format of a scope statement is given in Figure 3. The scope-spec is either a list of state names, T, or NIL. If the scope spec is a list of state names, then the action(s) can be executed when the parse begins at or has passed through any state in that list. If the scope-spec is T, the action is not to be executed until the parse has hypothesized the left end for that level of the grammar.

The following example from one of HWIM's grammars illustrates the notation for expressing scoping:

```
(S/WHAT-IS-IN-BUDGET
  (JUMP S/WHAT-IS-BUDGETED)
  (JUMP S/POP
    (SCOPE (S/WHAT-DO S/WHAT-HAVE)
      (VERIFY (GETR LEFT))
      (SETR SUBJ (NPBUILD))))))
```

From this state in HWIM's grammar, it is possible to jump to S/POP to end the utterance only if we have a question such as "WHAT IS LEFT IN THE BUDGET." In these cases the register LEFT will have been set earlier. If we are parsing right to left, we must know how far left we must carry the parse before making this test. The JUMP arc to S/POP indicates in the scope of the VERIFY action that the parse must have passed through either S/WHAT-DO or S/WHAT-HAVE.

3.3 Cascaded ATN's

One of the long standing problems in natural language understanding has been dealing with the interaction of syntactic and semantic information. Ways of achieving close interaction between syntax and semantics have traditionally involved writing semantic interpretation rules in 1-1 correspondence with phrase structure rules (e.g., Thompson [7]), writing "semantic grammars" that integrate syntactic and semantic constraints in a single grammar (e.g., Burton [3]), or writing ad hoc programs that combine such information in unformalized ways. The first approach requires as many syntactic rules as semantic rules, and hence is not really much different from the semantic grammar approach (this is the conventional way of defining semantics of programming languages). The third approach, of course, may yield some level of operational system, but does not usually shed any light on how such interaction should be organized, and is difficult to extend.

The semantic grammar approach, while effective, tends to miss generalizations and its results do not extend well to new domains. It misses syntactic generalizations, for example, by having to duplicate the syntactic information necessary to characterize the determiner structures of noun phrases for each of the different semantic kinds of noun phrase that can be accepted. Likewise, it tends to miss semantic generalizations by repeating the same semantic tests in various places in the grammar when a given semantic constituent can occur in various places in a sentence. HWIM's "pragmatic" grammar is an instance of the semantic grammar approach carried one more level, and thus gains its integration at the expense of modularity, transportability, and brevity.

Rusty Bobrow's RUS parser [1] is the first parser to my knowledge to make a clean separation between syntactic and semantic specification while gaining the benefit of early and incremental semantic filtering and maintaining the factoring advantages of an ATN. Its operation can be characterized by a generalization of ATN grammars that I have called cascaded ATN's (CATN's). A cascade of ATN's provides a way to reduce having to say the same thing multiple times or in multiple places, while providing efficiency comparable to a semantic grammar and at the same time maintaining a clean separation between syntactic and semantic levels of description. It is essentially a mechanism for permitting decomposition of an ATN grammar into an assembly of cooperating ATN's, each with its own characteristic domain of responsibility.

A CATN is essentially a sequence of ATN transducers with each successive machine taking input from the output of the previous one. Specifically, a CATN is a sequence of ordinary ATN's that include among the actions on their arcs an operation TRANSMIT, which transmits an element to the next machine in the sequence. The first machine in the cascade takes its input from the input sequence, and subsequent machines take their input from the TRANSMIT commands of the previous ones. The output of the final machine in the cascade is the output of the machine as a whole. The only feedback from later stages to earlier ones is a filtering function that causes paths of the nondeterministic computation to die if a later stage cannot accept the output of an earlier one.

The conception of cascaded ATN's actually arose from observing the interaction between the lexical retrieval component and the linguistic component of the HWIM speech understanding

system. HWIM's lexical retrieval component made use of a network that consumed successive phonemes from the output of an acoustic phonetic recognizer and grouped them into words. Because of phonological effects across word boundaries, this network could consume several phonemes that were part of the transition into the next word before determining that a given word was possibly present. At certain points, it would return a found word together with a node in the network at which matching should begin to find the next word (essentially a state remembering how much of the next word has already been consumed due to the phonological word boundary effect). This can be viewed as an ATN that consumes phonemes and transmits words as soon as it has enough evidence that the word is there.

The lexical retrieval component of HWIM can thus be viewed as an ATN whose output drives another ATN. This leads to the conception of a complete speech understanding system as a cascade of ATN's, one for acoustic phonetic recognition, one for lexical retrieval (word recognition), one for syntax, one for semantics, and one for subsequent discourse tracking and other pragmatic constraints.

3.4 Lexical Retrieval as an ATN

One of the difficult aspects of constructing a lexically-driven speech understanding system is providing a mechanism to efficiently determine what words in the lexicon match the input. At least for conventional serial machines, it is not acceptable to separately compare each word in a large vocabulary against the input string. However, by viewing the lexicon as an ATN grammar

of acceptable phoneme sequences, one can factor together the common parts of words that begin the same, so that the grammar begins from a single state with arcs for each phoneme that can begin a word leading to states that compactly represent the sequence of phonemes recognized so far and the set of phonemes that can follow them. This much structure, which is illustrated in Figure 4, is the same as a classical decision tree, and is effectively the same structure used by the CMU Harpy system to organize its entire recognition system. However, by viewing it as an ATN, it is possible to account for some more subtle phenomena.

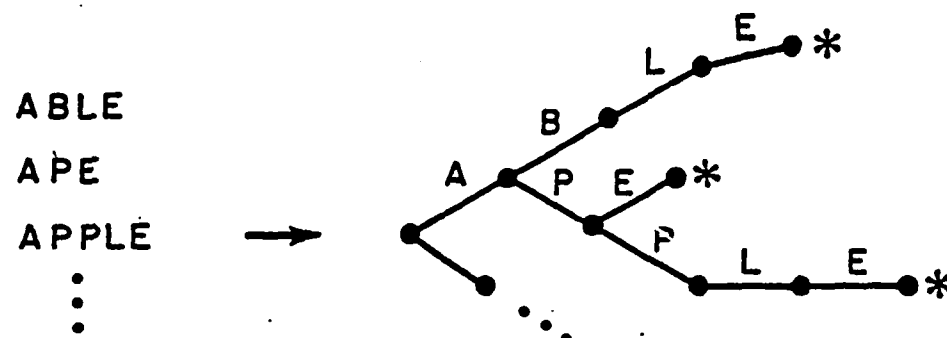


FIG. 4. A LEXICAL DECISION TREE

Specifically, by viewing the lexical component as a stage of an ATN cascade that accepts a sequence of phonemes and generates a sequence of words, it is possible to model the behavior of

Klovstad's lexical retrieval component in the HWIM system [11], whereby cross-word-boundary phonological rules are efficiently handled. Klovstad's technique consisted of wrapping such a lexical decision tree around on itself by allowing jump transitions from its leaves to its root, so that viewed as an ATN it accepted sequences of words rather than simply single words. Klovstad then matched cross-word-boundary phonological rules across these special jump transitions and spliced in the resulting changes in cross-word-boundary pronunciations from the point in the word where the change began to the point in the decision tree where the remainder of the next word would begin.

Klovstad was then able to merge common inter-word coarticulation patterns by remembering the word involved on the left (as if in an ATN register) before entering the portion of the network that was shared by different words, and he was able to confirm that word if he found the appropriate coarticulation pattern (at which point he would already have consumed an initial portion of the next word and be positioned at the right point in the discrimination net to continue recognizing whatever word it was). We can view this as transmitting the recognized word slightly out of synchrony with the consumption of input phonemes. The lexical retriever can often identify what word is expected before it gets all the evidence necessary to confirm its presence. When it has confirmed the presence of a word, it will sometimes have already begun to accumulate evidence for the next one. (This complexity of the word boundary phenomenon is one of the major difficulties that sets speech understanding apart from text understanding, where the boundaries between words are clearly marked by spaces and punctuation.)

3.5 Benefits of CATN's

The decomposition of a natural language analyzer into a cascade of ATN's gains a "factoring" advantage similar to that which ATN's themselves provide with respect to ordinary phrase structure grammars. Specifically, the cascading allows alternative configurations in the later stages of the cascade to share common processing in the earlier stages that would otherwise have to be done independently. That is, if several semantic hypotheses can use a certain kind of constituent at a given place, there need be only one syntactic process to recognize it.

Cascades also provide a simpler overall description of the acceptable input sequences than a single monolithic ATN combining all of the information into a single network would give. That is, if any semantic level process can use a certain kind of constituent at a given place, then there need be only one place in the syntactic stage ATN that will recognize it. Conversely, if there are several syntactic contexts in which a constituent filling a given semantic role can be found, there need be only one place in the semantic ATN to receive that role. (A single network covering the same facts would be expected to have a number of states on the order of the product, rather than the sum, of the numbers of states in the individual stages of the cascade.)

One might note here as an aside that an additional advantage provided by the factoring aspects of a cascade, for future systems that will learn much of their behavior, is the localization of activities in a single place where a given linguistic fact to be learned. Without such factoring,

essentially the same syntactic fact might have to be learned separately in different semantic contexts, and a given semantic fact might have to be learned separately in different syntactic contexts. Moreover, the separation of the stages of the cascade provides a decomposition of the overall problem into individually learnable skills. These facts may be of significance not only for theories of human language development and use, but also for computer systems that can be easily debugged or can contribute to their own acquisition of improved language skill. The above facts suggest that the traditional characterization of natural language in terms of the levels of phonemes, syllables, words, phrases, sentences, and higher level pragmatic constructs may be more than just a convenient way to present the subject.

4. MIDDLE OUT PARSING WITH ATN'S

HWIM's control strategy requires that the syntactic component be able to take any island (i.e., consecutive sequence of word matches) and determine if it can be parsed as an acceptable fragment of a sentence. If so, the syntactic component must be able to return a list of words and categories that would form acceptable extensions to the fragment at either end. The constraints this places on the parser are that it must be able to start at any point in the grammar and at any point in the input and work in either direction. Moreover, it must be able to process islands that may partially or fully traverse several different levels of the grammar. In this section we will describe a middle-out parsing system for ATN grammars that supports the island-driven control strategies of HWIM.

Whereas the state of a parse for a conventional left-to-right ATN can be represented by the current state, the register contents, and a stack of unfinished configurations at higher levels, the representation of a configuration for a middle-out parser is considerably more complex. Since in general one needs to represent the parsing of an island that contains incomplete constituents at each end, a representation is required that can deal with fragments such as "man saw the girl in" as in "The man saw the girl in the park." This in general requires a collection of partial analyses of segments at different levels whose overall structure is in the shape of a "stile" (a set of steps that goes over a fence). That is, there is some topmost fragment with (in general) an incomplete partial analysis at a lower level at both its left and its right, which can in turn have partial analyses at still lower levels adjacent to them, etc. The prototypical structure is illustrated in Figure 5.

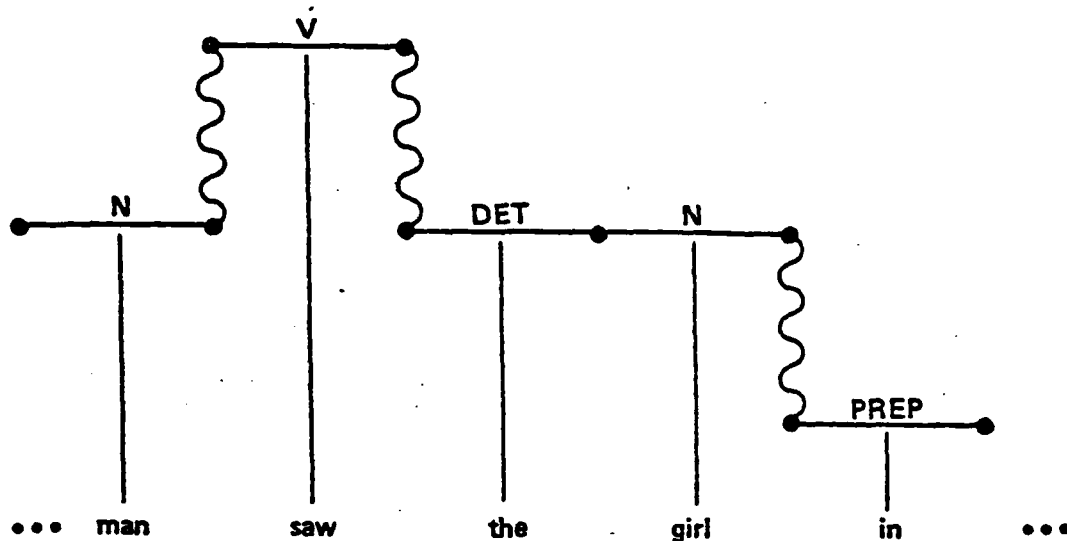


FIG. 5. A PROTOTYPICAL ISLAND CONFIGURATION IS SHAPED LIKE A STILE

The horizontal lines in Figure 5 correspond to transitions from one state to another annotated with the word in the input that enabled the transition. Sequences of such transitions at a single level will be called segment configurations (or SCONFIGS) and a collection of SCONFIGS covering an island (forming one or more stiles) will be called an island configuration (or ICONFIG). In general, an SCONFIG in an ICONFIG may be adjacent to several other SCONFIGS at other levels, and thus participate in several different stiles covering the island, corresponding to different ways to parse it. We will refer to each sequence of compatible SCONFIGS forming a stile covering the island as a path through the island. The parser will at various points need to trace paths through the island in order to determine which SCONFIGS at one end of the island are compatible with which SCONFIGS at the other.

The wavy vertical lines in Figure 5 correspond to relations of indirect pushing and popping in the ATN grammar. It is important that the relationships be indirect, so that the representation can be nonexplicit about the number of possible intervening levels of structure that might occur in connecting one such SCONFIG to another. This use of indirect pushing and popping relations is similar to the technique used in Earley's algorithm [4] to avoid the combinatorics associated with explicitly enumerating alternative stacks and has similar advantages here.

4.1 State Relations Used in the HWIM Parser

The HWIM parser makes use of a variety of indices constructed from its ATN grammar in order to efficiently perform its analyses. One of these is an index by word and syntactic category to all of the WRD and CAT arcs in the grammar, so that given a word, it can efficiently identify all of the arcs that can use that word. Another operation that it needs to perform efficiently is to identify whether an arc can be joined to an SCONFIG by a sequence of nonconsuming transitions (i.e., jumps, pushes, and pops). To support this it makes use of indices based on a set of relations between states that is in some sense a generalization of the "left closure" relationship used in Earley's algorithm, LR-k grammars and selective top-down and selective bottom-to-top parsing algorithms.

Let us define the relation B (for "begin") to hold between two states x and y if there is a push arc leaving state x that pushes to state y. Similarly we will define the relation C (for

"complete") to hold between states x and y if state x can pop a constituent that is consumed by a push arc whose destination is state y . That is, the relation B holds between states bridging the left boundary of a constituent and C holds between states bridging a right boundary. We will define the relation J to hold between x and y if there is a jump arc from state x to state y .

The relations B , C , and J cover all of the ways that an ATN can make transitions between states without consuming any input. Some generalized closure relations derived from these relations (and their converses, which we will call UB , UC , and LJ , for "unbegin", "uncomplete", and "left jump") play an important role in the HWIM parser. We use the Kleene star (*) to indicate 0 or more occurrences of a relation and the symbol $+$ to represent the disjunctive union of two relations. For example, the relation $(x J^* y)$ is true if x is the same state as y or if there exists some sequence of jump arcs leading from x to y . The relation $(x (J + P) y)$ is true if there is either a jump arc from x to y or if there is a push arc leaving state x that pushes to y . We can also concatenate these basic relationships to describe a sequence of transitions. For example, the relation $(x J^* B J^* y)$ is true if there are intermediate states z and w such that the relations $(x J^* z)$, $(z B w)$, and $(w J^* y)$ are all true.

Using this notation we can now name some additional relations that are important to the HWIM parser. These are:

$$\begin{aligned} B! &= B (J^* B)^* \\ C! &= C (J^* C)^* (J^* B)^* \\ UB! &= UB (LJ^* UB)^* (LJ^* UC)^* \\ UC! &= UC (LJ^* UC)^* \end{aligned}$$

The relation $B!$ corresponds to indirect pushing from the left, while $C!$ corresponds to going up and over a stile of

nonconsuming arcs by completing some phrases and then beginning some new ones. UB! corresponds to going over a stile from right to left, while UC! is the transition from higher to lower levels coming from the right (the left-hand version of a push). Even more useful than these relations are the relations $J*BIJ*$, $J*CIJ*$, $LJ*UB!LJ*$ and $LJ*UC!LJ*$. These can be intuitively described in English as the paths that will reach all possible states on other levels of the grammar that can be reached using only non-consuming arcs. The utility of these indices is fairly clear; if there is a segment configuration ending in state x , and a new word is to be added to the island that can be consumed by an arc that begins at state y , then there can be a path that will connect the island to the new word only if the relation $(x (J* + J*BIJ* + J*CIJ*) y)$ is true.

4.2 Paths

Two adjacent SCONFIGS in an ICONFIG will be said to be compatible if the left one stands in the $(J*BIJ* + J*CIJ*)$ relation to the right one (i.e., the right state of the left one stands in that relation to the left state of the right one). That is, two adjacent segments are compatible if the left one either pushes or pops, perhaps indirectly, to the right one. A sequence of compatible adjacent segments will be called a path provided that it contains no sequence of three segments such that the first stands in the $J*BIJ*$ relation to the second, which in turn stands in the $J*CIJ*$ relation to the third. (Whenever such a compatible sequence of three segments exists, the middle one can be completed and consumed by a PUSH arc that incorporates it into either the left or right segment or perhaps combines all three into a single segment.)

Associated with each path is an indication of which segment is at a higher level in the grammar than any other. Note that it is possible to have two paths that are identical except for which segment is chosen as the top. Marking the top segment of a path divides the remaining ones into two groups, which must then be at successively lower levels as they get further away from the top.

Paths are used by the HWIM parser for making predictions adjacent to an island. In order to add a new word to an island, it may be necessary to complete one of the SCONFIGs at that end and pop its constituent to one of the internal SCONFIGs, which will then reach the end of the island and connect to the new word. The internal SCONFIGs that need to be considered are on the paths leading from the SCONFIG being completed. Also, when a new word is added to one end of an island, it may be incompatible with some of the SCONFIGs at that end. These incompatible SCONFIGs are purged from the representation of the resulting new island. However, their removal may leave other internal SCONFIGs "disconnected" in that they do not now participate in any paths through the island. All such SCONFIGs are purged from the representation. Ultimately this may result in the removal of SCONFIGs at the other end of the island. In this way, words added at one end of an island can result in tightening the predictions of compatible words at the other end.

4.3 The Parser

The HWIM parser has four basic actions, corresponding to different tasks it is called upon to perform by the Control component. The first, seed event processing, creates a new one-

word island and a representation for every path covering that word that the grammar allows (i.e., each arc in the grammar that can consume it). The second, word event processing, adds a new word to one end of an existing island, extending those paths that allow the new word and eliminating those SCONFIGs that do not participate in any such path. The third action of the parser is end event processing. This takes place when an existing island has reached one end of the utterance. The parser then extends the paths that can reach the start state of the grammar (for a left end event) or a final state of the grammar (for a right end event) without consuming any additional words. If the other end of the island is open, the parser returns a set of predictions at that end, which may be more restricted than before; if that end is also complete, the parser returns a complete parse.

The fourth type of action, island collision event processing allows the Control component to combine two islands with a one-word gap between them and a new word filling that gap. Although one can develop special techniques for combining the parses of two islands, it is sufficient to simply perform new word events until one island has incorporated all the words of the other. This is how island collision events were implemented in the HWIM parser, for reasons of expediency.

4.4 The Principal Parser Functions

In the remainder of this section we will describe the main functions in the HWIM parser and their operation. The presentation is intended to reveal some of the complexities of the middle-out parsing algorithm. There is not enough detail to

guide an implementation, although with diligent study of the section that might be possible. However, the discussion is pretty demanding, so the casual reader may well want to skim it or skip to the next section.

The functions described here are all suffixed -RIGHT to indicate that they are used when adding a new word to the right of an existing island. There is a set of similar functions (suffixed -LEFT), which work on adding words to the left. The differences between the left-to-right and right-to-left functions are small, brought on by the fact that a new level is begun at one particular state on the left (the label of the PUSH arc) but can pop from any of a number of accepting states (states with POP arcs) on the right. These differences usually involve only an extra loop to be executed a bounded number of times.

4.5 Starting an Island

To begin a one-word island, the parser finds all arcs in the grammar that can consume the new word match, using an index of pointers from each word and category to the arcs that consume them. It processes those arcs, creating a segment configuration (SCONFIG) for each one. These are collected into an island configuration (ICONFIG), which is then processed to find the proposals to return to Control.

4.6 Processing an Arc

The basic transition function in the parser, DOARC, takes an

old SCONFIG, an adjacent arc, and, if it is a consuming arc, a word or constituent to be consumed. The result is a new SCONFIG representing the state of the computation after the new arc and input have been incorporated. DOARC is applied to a list of SCONFIGs to be processed for a given arc and direction. The result is a new list of SCONFIGs. Its operation is as follows:

For each SCONFIG do:

- a) Verify that the arc is adjacent to the old SCONFIG;
- b) Do the context free actions on the arc;
- c) For each context sensitive action on the arc: - if the context is not yet complete, create an undone scoped action to be added to the new SCONFIG - otherwise evaluate the action.
- d) If the direction is to the left, see if the scope of any saved actions in the old SCONFIG has now been satisfied. If so, evaluate those actions.
- e) Construct the new SCONFIG by: - setting the new boundaries and end states - saving the registers and undone scoped actions - adding the left state of the arc to the list of states that the computation has passed through (if the direction is to the left).

4.7 Connecting a New Word to an Island

In extending a segment at the end of an old island to meet a new word to be added, the sequence of intervening non-consuming arcs that make the connection can be all at the same level of the network (the J* case); they can change to a lower level (the J*B!J* case); or they can change levels in a way equivalent to one or more POPs followed by zero or more PUSHes (the J*C!J*

case). The function CONNECT-RIGHT takes groups of SCONFIGs that terminate at the right boundary of an existing island and joins them to sets of arcs that can consume the new word to be added. We will call these arcs that can consume the new word that CONNECT-RIGHT is trying to connect to the island the destination arcs. CONNECT-RIGHT calls a function EXTEND-PATHS-RIGHT to do most of the work.

EXTEND-PATHS-RIGHT follows jump transitions to states that either 1) begin a destination arc, 2) push for a constituent that contains a destination arc, or 3) pop from the current constituent to a higher level that contains (or can then push for) a destination arc. EXTEND-PATHS-RIGHT begins from an SCONFIG group whose members share the same right boundary and completes the JUMP paths on that level that are required to reach the states mentioned above.

EXTEND-PATHS-RIGHT is used in several contexts in CONNECT-RIGHT, so it must have a way of deciding which of the above cases are relevant. To do this it must be supplied with an argument (TRYDIRS) indicating how many of the above three ways to look for a connection. The operation of EXTEND-PATHS-RIGHT can be described as follows:

Let $J^*TOSTATES$ be the set of states that can J^* to one of the destination arcs. Let $J^*FROMSTATES$ be the union of the J^* sets of the right end states of the input SCONFIGs.

If the value of TRYDIRS is 1 or more (i.e., always), EXTEND-PATHS-RIGHT considers states on the same level as the input SCONFIGs (i.e., the states in $J^*FROMSTATES$). All possible jump paths that reach one of the destination arcs are processed by DOARC. The resulting SCONFIGs are collected into groups

according to their rightmost state and extended to include the destination arcs to which they attach.

In addition, if the value of TRYDIRS is 2 or more, then EXTEND-PATHS-RIGHT considers paths that lead to a lower level constituent, as well as ones on the same level as the input SCONFIGs. Each of the states x in $J*FROMSTATES$ is also tested to see if it can reach a state y in $J*TOSTATES$ via a $B!$ transition. If so, then all jump paths are completed from the input SCONFIGs to state x , and new SCONFIGs starting in state y are created and given EXTEND-PATHS-RIGHT with TRYDIRS = 1.

In addition, if the number of directions (TRYDIRS) is 3, EXTEND-PATHS-RIGHT also considers paths that complete segments of the input SCONFIGs and return to a higher level. These higher levels may contain one of the destination arcs or may then push for new constituents that contain a destination arc. Note that these paths are just the $C!$ set. For each state x in $J*FROMSTATES$ that can reach a state in $J*TOSTATES$ via a $C!$ transition, all jump paths from the input SCONFIGs to x are created, and the resulting group of SCONFIGs is placed in a completion queue $C!Q$.

4.8 Processing the Completion Queue

The $C!Q$ queue contains all SCONFIGs that have reached an accepting state. However, there is a major difference between those of its segments which are now complete at both ends and those which are still incomplete at the other end. In the first case, the appropriate constituent must be created and the process resumed at the higher level (COMPLETE-RIGHT). In the second

case, all states in the C! set of the right state of the segment must still be considered for extension, since the segment has no left context.

SPLITC!Q checks the left boundary of each SCONFIG in the C!Q. If it is the left boundary of the island, then the segment is considered open and is put in the C!OPENLEFTQ. If it is within the island, then SPLITC!Q picks up RSTATES, the list of right end states, from the list of segments whose right end coincides with the left end of the segment being considered. This is done by keeping an index of segments in the island by right boundary. If there is an intersection between RSTATES and the UB! set of the left end state of the SCONFIG being considered, then the SCONFIG could have been pushed for from one of those segments, so the SCONFIG is put in the C!COMPLETEQ. If there is an intersection between RSTATES and the UC! set of the left end states of the SCONFIG, then it is possible for the SCONFIG to belong to a path in which it is the highest level SCONFIG. It is therefore put in the C!OPENLEFTQ. It is possible for an SCONFIG to be in both queues as a result of different possible paths.

At the end of this process, C!OPENLEFTQ is given to EXTEND-PATHS-RIGHT with TRYDIR = 2, and C!COMPLETEQ is given to a function COMPLETE-RIGHT.

4.9 Completing a Constituent

The function COMPLETE-RIGHT is called to build a constituent for a segment that can connect to a destination arc by popping and which is pushed for (perhaps indirectly) by an SCONFIG to the left in the island. COMPLETE-RIGHT creates the new constituent

to be popped, joining it to the island in one of two ways. It may add it to an existing segment at the next higher level, or if such a segment does not exist, it will create a new intermediate segment for each arc that can use the completed constituent and is compatible with some SCONFIG of the left.

When completing a segment and creating a new constituent, COMPLETE-RIGHT causes all undone scoped actions to be executed and creates the new constituent from the label of the POP arc. COMPLETE-RIGHT then looks to see if any existing segment immediately to the left of the new constituent could have pushed for it. If so, its PUSH arc is executed to include the new constituent. COMPLETE-RIGHT also looks to see if any segment to the left of the new constituent can push for it indirectly and still reach a destination arc. For each of these cases, COMPLETE-RIGHT must create a new segment, extend it from its beginning state to the state that pushes for the new constituent, and execute that push arc to create a new segment that is complete on the left and that includes the new constituent on the right.

After a segment has been extended to include the new constituent, it is placed back in the TODOQ because it may need to be further extended. CONNECT-RIGHT will deal with this correctly since the new segment behaves very much like the segments that were in the old island.

4.10 Making Predictions

PREDICT-RIGHT is a function that creates a list of all terminal-consuming arcs that can be reached by any path of non-

consuming arcs from the right end of an island. The word and category predictions made by the Syntactic component are collected from this list and a corresponding one for the left end of the island.

It is not sufficient for PREDICT-RIGHT to merely list all arcs that can be reached from states that can end the island; these predictions must be restricted by segment configurations within the island that are at a higher level than segments on the right. To make the list of predicted arcs, PREDICT-RIGHT groups the SCONFIGs by end states and boundaries so that ambiguous parses of one level are together. For each SCONFIG group at the right boundary, PREDICT-RIGHT checks to see if it can reach the left end of the island without passing through a higher level segment. If so, then all consuming arcs in all states reachable by $(J^* + J^*B!J^* + J^*C!J^*)$ are predicted. If not, PREDICT-RIGHT divides the predictions into two cases. First, all terminal consuming arcs in states reachable from the SCONFIG by $(J^* + J^*B!J^*)$ are predicted. Secondly, if the SCONFIG has a nonempty $J^*C!$ set, then for each SCONFIG to its left that can push for it, PREDICT-RIGHT repeats the prediction process on the higher level SCONFIG as if its right end state were the right state of the PUSH arc that pushes for the segment at the right boundary. As a result of this process, all predictions on the right that are compatible with possible paths across the island are generated.

5. MIDDLE-OUT PARSING WITH CATN'S

[13] presents a discussion of parsing with CATN's in a conventional left-to-right mode. For our purposes, however, it is interesting to consider the problem of a middle-out algorithm. Since the breakdown that HWIM imposes between the lexical retrieval component and the linguistic consultant is exactly the boundary between two stages of an ATN cascade when viewed appropriately, the island-driven control strategy coupled with the middle-out ATN parser described above can be viewed as an instance of a CATN parser.

This view of the HWIM system as a middle-out parser for an ATN cascade suggests an approach to a minor problem that was present in the HWIM implementation. HWIM's lexical retrieval component used two dictionary trees, one going left-to-right and one going right-to-left, in order to support growing an island in either direction. This technique did not really deal adequately with seed words, which were handled by grouping both left-to-right and right-to-left matches of the same word into a single "fuzzy word match" (see [14]). Left-to-right seeds could handle inter-word coarticulation on the right, and right-to-left seeds could handle such coarticulation on the left, but there was no mechanism for forming seeds with coarticulation effects at both ends. The best one could do in such cases was a fuzzy word match with some of its elements faithfully representing the left end effects and some representing the right end effects. Neither of these would have an ideal score because of their failure to model the coarticulation effect at their other end.

If one used the same approach in the lexical ATN that is used in the middle out parsing of the linguistic component, then

one could begin word match hypotheses from highly reliable seed hypotheses such as vowels and sonorants in stressed syllables and then extend such hypotheses in both directions. This would in fact be considerably simpler than for the syntactic parser, since the lexical ATN contains no pushing and popping. An ICONFIG for the lexical ATN would consist of merely a single SCONFIG recording: the left and right end states in the lexical ATN, the score of the match, and possibly a register remembering the word to be transmitted or an undone scoped action to set such a register when a suitable left state is encountered. This approach would eliminate the need for two separate dictionary trees, and would create possibilities for somewhat different factoring of the common parts of the dictionary. (Whereas the HWIM dictionary tree merges common initial portions of word pronunciations, this approach would permit merging designed to capture commonalities on both left and right.)

Perhaps more interestingly, one could introduce a separate stage of the cascade for a syllable level analysis, with seed events starting at syllable nuclei and moving outward in both directions until they encounter adjacent syllables via island collision. This would have the advantage of moving from reliable acoustic phonetic evidence into less reliable regions in exactly the way that a theoretical analysis of the shortfall density algorithm shows to be effective.

Another application of cascades would be to replace HWIM's pragmatic grammar with a cascade of separate ATN's for syntax, semantics, and pragmatics. This would provide a cleaner separation between the different sources of knowledge and an overall reduction in the size of the grammar needed to capture a comprehensive range of interactions between syntactic, semantic,

and pragmatic phenomena. Further developments along these lines would include cascades of generalized transition networks [12] and the use of sophisticated knowledge representation systems such as KL-ONE [2, 15].

ACKNOWLEDGMENTS

The speech understanding system HWIM is the result of a large group effort. Participants in the project included Madeleine Bates, Geoffrey Brown, Bertram Bruce, Craig Cook, Laura Gould, Gregory Harris, Dennis Klatt, Jack Klovstad, John Makhoul, Bonnie Webber, Richard Schwartz, and Victor Zue. The middle-out parser described here was implemented by Geoffrey Brown. The initial speech research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Office of Naval Research under contract N00014-75-C-0053. Subsequent theoretical work and the writing of this report was supported in part by ONR under Contract N00014-77-C-0371.

REFERENCES

- [1] Bobrow, R.J.
The RUS System.
BBN Report 3878, Bolt Beranek and Newman Inc., 1978.
- [2] Brachman, R.J. and Schmolze, J.
An Overview of the KL-ONE Knowledge Representation System.
Cognitive Science, 1984.
- [3] Burton, R.R.
Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems.
BBN Report No. 3453, Bolt Beranek and Newman Inc., December, 1976.
- [4] Earley, J.
An Efficient Context-Free Parsing Algorithm.
Ph.D. Dissertation, Computer Science Dept., Carnegie-Mellon University, August, 1968.
- [5] Lesser, V.R., Fennell, R.D., Erman, L.D., and Reddy, D.R.
Organization of the Hearsay II Speech Understanding System.
IEEE Trans. Acoustics, Speech, and Signal Processing ASSP-23(1):11-24, 1975.
- [6] Lowerre, B.T.
The HARPY Speech Recognition System.
Technical Report, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1976.
- [7] Thompson, F.B.
The Semantic Interface in Man-Machine Communication.
Technical Report RM 63TMP-35, General Electric Co., Santa Barbara, September, 1963.
- [8] Wolf, J.J. and Woods, W.A.
The HWIM Speech Understanding System.
In Wayne A. Lea (editor), Trends in Speech Recognition, pages 1-24. Prentice-Hall, Inc., Englewood Cliffs, NJ, 1980.
- [9] Woods, W.A.
Transition Network Grammars for Natural Language Analysis.
CACM 13(10):591-606, October, 1970.
- [10] Woods, W.A., Bates, M., Brown, G., Bruce, B., Cook, C.,

- Klovstad, J., Makhoul, J., Nash-Webber, B., Schwartz, R., Wolf, J., Zue, V.
Speech Understanding Systems, Final Technical Progress Report, Volumes I-V.
Technical Report 3848, BBN, 1976.
- [11] Woods, W.A., Bates, M., Brown, G., Bruce, B., Cook, C., Klovstad, J., Makhoul, J., Nash-Webber, B., Schwartz, R., Wolf, J., Zue, V.
Volume III, Lexicon, Lexical Retrieval and Control, Speech Understanding Systems, Final Technical Progress Report, 30 October 1974 to 29 October 1976.
Technical Report, BBN, 1976.
This volume may be obtained through NTIS by specifying AD No. AO35277.
- [12] Woods, W.A.
Research in Natural Language Understanding, Quarterly Progress Report No. 4: Generalizations of ATN Grammars.
BBNREP 3963, BBN, August, 1978.
- [13] Woods, W.A.
Cascaded ATN Grammars.
Amer. J. Computational Linguistics 6(1):1-15, Jan.-Mar., 1980.
- [14] Woods, W.A.
Optimal Search Strategies for Speech Understanding Control.
In Webber, B. and Nilsson N. (editors), Readings in Artificial Intelligence, pages 30-68. Tioga Publishing Co., Palo Alto, CA, 1981.
- [15] Woods, W.A.
What's Important About Knowledge Representation.
IEEE Computer Magazine, Special Issue on Knowledge Representation, September, 1983.

Official Distribution List

Contract N00014-77-C-0371

	<u>Copies</u>
Defense Documentation Center Cameron Station Alexandria, VA 22314	12
Office of Naval Research Information Systems Program Code 437 Arlington, VA 22217	2
Office of Naval Research Code 200 Arlington, VA 22217	1
Office of Naval Research Code 455 Arlington, VA 22217	1
Office of Naval Research Code 458 Arlington, VA 22217	1
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, MA 02210	1
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, IL 60605	1
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, CA 91106	1
Naval Research Laboratory Technical Information Division Code 2627 Washington, D.C. 20380	6

cont'd.

Naval Ocean Systems Center Advanced Software Technology Division Code 5200 San Diego, CA 92152	1
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1
Mr. E. H. Gleissner Naval Ship Research & Development Ctr. Computation & Mathematics Dept. Bethesda, MD 20084	1
Capt. Grace M. Hopper, USNR Naval Data Automation Command Code 00H Washington Navy Yard Washington, D.C. 20374	1
Mr. Paul M. Robinson, Jr. NAVDAC 33 Washington Navy Yard Washington, D.C. 20374	1
Advanced Research Projects Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, VA 22209	1
Capt. Richard L. Martin, USN 507 Breezy Point Crescent Norfolk, VA 23511	1
Director, National Security Agency Attn: R54, Mr. Page Fort G.G. Meade, MD 20755	1
Director, National Security Agency Attn: R54, Mr. Glick Fort G.G. Meade, MD 20755	1
Major James R. Kreer Chief, Information Sciences Dept. of the Air Force Air Force Office of Scientific Research European Office of Aerospace Research & Development Box 14 FPO New York 09510	1

cont'd.

Mr. Fred M. Griffiee
Technical Advisor C3 Division
Marine Corps Development
& Education Command
Quantico, VA 22134